College of Engineering, Construction & Living Sciences
Bachelor of Information Technology
ID721001: Mobile Application Development
Level 7, Credits 15

# Practicals

## Assessment Overview

In this assessment, you will solve 16 coding problems using **Kotlin** & **Kotlin command-line compiler**. This assessment contributes 15% towards your final mark in **IN721: Mobile Application Development**.

## Learning Outcomes

At the successful completion of this course, learners will be able to:

1. Implement & publish complete, non-trivial, industry-standard mobile applications following sound architectural & code-quality standards.
2. Identify relevant use cases for a mobile computing scenario & incorporate them into an effective user experience design.
3. Follow industry standard software engineering practice in the design of mobile applications.

## Assessment Table

| Assessment Activity | Weighting | Learning Outcomes | Assessment Grading Scheme | Completion Requirements |
|---|---|---|---|---|
| Project | 65% | 1, 2, 3 | CRA | Cumulative |
| Practicals | 15% | 1, 2, 3 | CRA | Cumulative |
| Presentation | 20% | 2, 3 | CRA | Cumulative |

## Conditions of Assessment

You will complete this assessment during your learner managed time, however, there will be availability during the weekly meetings to discuss the requirements & your progress of this assessment. This assessment will need to be completed by **Tuesday, 03 March 2022** at **5 PM**.

## Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **ID721001: Mobile Application Development**.

## Authenticity

All parts of your submitted assessment **must** be completely your work & any references **must** be cited appropriately including, externally-sourced graphic elements. Provide your references in a **README.md** file. All media **must** be royalty free (or legally purchased) for educational use. Failure to do this will result in a mark of **zero** for this assessment.

## Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning submissions, extensions, resubmissions & resits complies with **Otago Polytechnic** policies. Learners can view policies on the **Otago Polytechnic** website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

## Submission

You must submit all program files via **GitHub Classroom**. Here is the URL to the repository you will use for your code review – https://classroom.github.com/a/rWCfXF_l. Once you have completed this activity, create a pull request & assign the **GitHub** user **grayson-orr** to a reviewer. **Do not** merge your pull request.

## Extensions

Familiarise yourself with the assessment due date. If you need an extension, contact the course lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

## Resubmissions

Learners may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are to be completed within a negotiable short time frame & usually **must** be completed within the timing of the course to which the assessment relates. Resubmissions will be available to learners who have made a genuine attempt at the first assessment opportunity & achieved a **D grade (40-49%)**. The maximum grade awarded for resubmission will be **C-**.

## Resits

Resits & reassessments **are not** applicable in **ID721001: Mobile Application Development**.

## Instructions - Learning Outcomes 2, 3

Create a file for each problem.

### Problem 1 (0.5%):

Calculate the average of the given **double array** & display the expected output.

```
fun main() {
    val nums = doubleArrayOf(45.3, 67.5, -45.6, 20.34, -33.0, 45.6)

    // Write your solution here

    // Expected output:
    // Average: 16.69
}
```

## Problem 2 (0.5%):

Write a function called **fizzBuzz** which accepts an **Int** parameter called **num**. If **num** is a multiple of three, return **Fizz**, if **num** is a multiple of five, return **Buzz** & if **num** is a multiple of three & five, return **FizzBuzz**. Call the **fizzBuzz** function in the **main** function to display the expected output.

```
// Write your fizzBuzz function here

fun main() {
    for (i in 1..15 step 2) {
      // Write your solution here
    }

    // Expected output:
    // 1
    // Fizz
    // Buzz
    // 7
    // Fizz
    // 11
    // 13
    // FizzBuzz
}
```

## Problem 3 (0.5%):

You have been given two **mutable lists** containing the lecturer's favourite programming languages. Use the following hints to display the expected output:

- Add a specified element to the end of a list.
- Add all elements of a specified collection to the end of a list.
- If present, remove a specified element from a collection.
- Capitalise the element in the 3rd index.

```
fun main() {
    val progLangsOne: MutableList<String> = mutableListOf("C#", "JavaScript", "Kotlin", "OCaml")
    val progLangsTwo: MutableList<String> = mutableListOf("C++", "Go", "Swift", "TypeScript")

    // Write your solution here

    // Expected output:
    // [C#, JavaScript, Kotlin, OCAML, Prolog, C++, Swift]
}
```

## Problem 4 (0.5%):

You have been given a **mutable map** containing three soft drinks & their prices. Use the following hints and **Kotlin** aggregate operations to display the expected output:

- Change the price of Coca-Cola to 4.50.
- Calculate the total price of all soft drinks.

```
fun main() {
    val softDrinks: MutableMap<String, Double>
        = mutableMapOf("Coca-Cola" to 2.00, "Fanta" to 0.90, "Sprite" to 1.10)

    // Write your solution here

    // Expected output:
    // Total price: $6.50
}
```

## Problem 5 (0.5%):

You have been given two **mutable sets** containing two lecturer's course codes. Use the following hints to display the expected output:

- Return a set containing all elements that are contained by both collections.
- Return a set containing all distinct elements from both collections.

```
fun main() {
    val courseCodesOne: MutableSet<String> = mutableSetOf("IN607", "IN721", "IN728", "IN732")
    val courseCodesTwo: MutableSet<String> = mutableSetOf("IN512", "IN607", "IN728", "IN732")

    // Write your solution here

    // Expected output:
    // [IN607, IN728, IN732]
    // [IN607, IN721, IN728, IN732, IN512]
}
```

## Problem 6 (0.5%):

You have been given a 5x5 grid or a **2D array** of zeros. Use the appropriate construct(s)/range(s) to access the items in the grid, i.e., zeros & replace them with Xs.

```
fun main() {
    var seating = arrayOf<Array<Any>>()
    for (i in 0..4) {
        var seat = arrayOf<Any>()
        for (j in 0..4) {
            seat += 0
        }
        seating += seat
    }

    // Write your solution here

    for (seat in seating) {
        for (value in seat) {
```

```
            print("$value ")
        }
        println()
    }

    // Expected output:
    // 0 0 0 0 X
    // 0 0 0 0 0
    // X X X 0 X
    // 0 0 0 0 0
    // 0 0 0 0 X
}
```

## Problem 7 (0.5%):

In the expected output below, the staircase is of size three. Its base & height are both equal to **numOfSteps**. Also, it is drawn using the hash symbol. Write the logic in the **generateSteps** function in order to display the expected output.

```
fun generateSteps(numOfSteps: Int): MutableList<String> {
    val stepSeq: MutableList<String> = mutableListOf()

    // Write your solution here

    return stepSeq
}

fun main() {
    for (step in generateSteps(4)) {
        // Expected output:
        println(step) // #
                      // ##
                      // ###
                      // ####
    }
}
```

## Problem 8 (0.5%):

You have been given a function called **defangAddress** which accepts a **String** parameter called **address**. This function returns a defanged version of **address**. A defanged address replaces every period **"."** with **"[.]"**. Write the logic in the **defangAddress** function in order to display the expected output.

```
fun defangAddress(address: String): String {
    var defangedAddr = ""

    // Write your solution here

    return defangedAddr
}

fun main() {
    // Expected output:
    println(defangAddress("255.100.50.0")) // 255[.]100[.]50[.]0
}
```

## Problem 9 (0.5%):

You have been given an incomplete function called **isPerfectNumber** which accepts an **Int** parameter called **num**. If **num** is a perfect number, return **true**, otherwise return **false**. A perfect num is a positive integer that is equal to the sum of its positive divisors excluding the number itself.

```
// Example 1
Input: num = 6
Output: true

// Example 2
Input: num = 2
Output: false

fun isPerfectNumber(num: Int): Boolean {
    // Write your solution here
}

fun main() {
    // Expected output:
    println(isPerfectNumber(5)) // false
    println(isPerfectNumber(6)) // true
}
```

## Problem 10 (0.5%):

You have been given an incomplete function called **removeDuplicates** which accepts an **IntArray** parameter called **nums**. Given a sorted **integer array,** remove the duplicates such that each element occurs only once & return the new length of the **array**.

```
fun removeDuplicates(nums: IntArray): Int {
    // Write your solution here
}

fun main() {
    // Expected output:
    println(removeDuplicates(intArrayOf(0, 0, 1, 1, 2, 2, 3, 3, 4))) // 5
}
```

## Problem 11 (1%):

Write two classes called **SoftwareDeveloper** & **Manager** which inherit from the given **Employee** class. The **SoftwareDeveloper** class has one additional class property called **favProgLang** of type **String**. The **Manager** class also has one additional class property called **employees** of type **MutableList<Employee>** & three functions which add, remove & display all managed employees.

Use the three **SoftwareDeveloper** objects & **Manager** object in the **main** function to display the expected output.

```
open class Employee(var id: Int, val firstName: String, val lastName: String, val salary: Int) {
    override fun toString() = "${firstName} ${lastName}"
}

// Write your SoftwareDeveloper class here

// Write your Manager class here
```

```
fun main() {
    val sftDevOne = SoftwareDeveloper(1, "Bert", "Watts", 100000, "Cobol")
    val sftDevTwo = SoftwareDeveloper(2, "Sara", "Cain", 75000, "Perl")
    val sftDevThree = SoftwareDeveloper(3, "Samantha", "Baker", 75000, "PHP")
    val manager = Manager(4, "Owen", "James", 150000, mutableListOf(sftDevOne, sftDevTwo))

    // Write your solution here

    // Expected output:
    // Sara Cain
    // Samantha Baker
}
```

## Problem 12 (1%):

You have been given a class called **Stack** of type **String**. Use the **Stack** object in the **main** function to display the expected output.

```
class Stack<String>() {
    private val els = mutableListOf<String>()
    fun push(el: String) = els.add(el)
    fun peek(): String = els.last()
    fun pop(): String = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun main() {
    val stack: Stack<String> = Stack()
    stack.push("Django")
    stack.push("Laravel")
    stack.push("Ruby on Rails")
    stack.push("Spring")

    // Write your solution here

    // Expected output:
    // Stack[Django, Laravel, Ruby on Rails]
    // Ruby on Rails is at the top of the stack
    // There are 3 item(s) in the stack
}
```

## Problem 13 (1%):

You have been given a class called **Stack** of type **String**. Use the **Stack** object in the **main** function & the **readLine** function to reverse the user's input.

```
class Stack<String>() {
    private val els = mutableListOf<String>()
    fun push(el: String) = els.add(el)
    fun peek(): String = els.last()
    fun pop(): String = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
```

```
}

fun main() {
    val stack: Stack<String> = Stack()

    // Write your solution here

    // Expected output:
    // Enter some text: John Doe
    // eoD nhoJ
}
```

## Problem 14 (1%):

You have been given a class called **Stack** of type **Int**. Use the **Stack** object in the **main** function & the **readLine** function to convert the user's input into binary.

```
class Stack<Int>() {
    private val els = mutableListOf<Int>()
    fun push(el: Int) = els.add(el)
    fun peek(): Int = els.last()
    fun pop(): Int = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun main() {
    val stack: Stack<Int> = Stack()

    // Write your solution here

    // Expected output:
    // Enter a number: 50
    // 110010
}
```

## Problem 15 (1%):

You have been given a class called **Stack** of type **Char** & an incomplete function called **isBalanced** which accepts a **String** parameter called **sequence**. Given a **sequence** containing only parentheses, curly brackets & square brackets, determine if **sequence** is valid.

```
class Stack<Char>() {
    private val els = mutableListOf<Char>()
    fun push(el: Char) = els.add(el)
    fun peek(): Char = els.last()
    fun pop(): Char = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun isBalanced(sequence: String): Boolean {
    val stack: Stack<Char> = Stack()
    val map = mapOf(
```

```
        '(' to ')', ')' to '(',
        '[' to ']', ']' to '[',
        '{' to '}', '}' to '{'
    )

    // Write your solution here
}

fun main() {
    // Expected output:
    println(isBalanced("{([])}")) // true
    println(isBalanced("{(["))  // false
}
```

**sequence** is valid if:

- Open bracket must be closed by the same bracket type.
- Open bracket must be closed in the correct order.

```
// Example 1
Input: sequence = "()"
Output: true
// Example 2
Input: sequence = "()[]{}"
Output: true
// Example 3
Input: sequence = "{]"
Output: false
// Example 4
Input: sequence = "{[}]"
Output: false
```

## Problem 16 (5%):

In this problem, you are going to build a **Rock, Paper, Scissors** game.

Create a function which simulates the **Rock, Paper, Scissors** game. The function takes the input of both players (rock, paper or scissors), first parameter from the first player, second from the second player. The function returns the result as such:

- First player wins
- Second player wins
- Draw

```
// Examples
rockPaperScissor("paper", "rock") => "First player wins"
rockPaperScissor("rock", "paper") => "Second player wins"
rockPaperScissor("paper", "paper") => "Draw"
```